PATENT
PDNO 10006298-1

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Application of: Amir Said	: Confirmation No. 5635
Serial No. 09/912,278	: Examiner: Colin M. Larose
Filed: July 24, 2001	: Group Art Unit: 2623

For: **IMAGE BLOCK CLASSIFICATION BASED ON ENTROPY OF DIFFERENCES**

RULE 131 DECLARATION BY INVENTOR

I, the undersigned, declare that:

1. I am the sole inventor in the above-captioned patent application.

2. I prepared a technical report entitled "Classification method for compound document segmentation and compression" and dated September, 1999 (the "Technical Report"). The Technical Report was prepared for the internal use of the Hewlett-Packard Company. Reference number of the Technical Report is HPL-1999-114. A copy of the Technical Report is attached.

3. I prepared an invention disclosure entitled "Block classification based on entropy of differences" (the "Invention Disclosure"). A copy of the Invention Disclosure is attached. The Invention Disclosure makes reference to the Technical Report as HPL-1999-114. I signed and dated the Invention Disclosure on August 4, 2000 and submitted the Invention Disclosure to the intellectual property department of the Hewlett-Packard Company.

4. All statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

SEPTEMBER 3, 2004
Date

Amir Said

Write in Dark Ink on Front Side Only, Please

INVENTION DISCLOSURE

PAGE ONE OF

PDNO 1006298 DATE RCVD 8/7/00 ATTORNEY SEH

*Instructions: The information contained in this document is **COMPANY CONFIDENTIAL** and may not be disclosed to others without prior authorization. Submit this disclosure to the HP Legal Department as soon as possible. No patent protection is possible until a patent application is authorized, prepared, and submitted to the Government.*

Descriptive Title of Invention:
IMAGE BLOCK CLASSIFICATION BASED ON ENTROPY OF DIFFERENCES

Name of Project:
COMPRESSION AND MULTIMEDIA TECHNOLOGIES

Product Name or Number:

Was a description of the invention published, or are you planning to publish? If so, the date(s) and publication(s):

NO

Was a product including the invention announced, offered for sale, sold, or is such activity proposed? If so, the date(s) and location(s):

NO

Was the invention disclosed to anyone outside of HP, or will such disclosure occur? If so, the date(s) and name(s):

NO

If any of the above situations will occur within 3 months, call your IP attorney or the Legal Department now at 1-888-4919 or 970-898-4919.

Was the invention described in a lab book or other record? If so, please identify (lab book #, etc.)

YES. REPORT HPL 1999-114

Was the invention built or tested? If so, the date:

YES, 5/99 (BLADE, VERSION 1.0)

Was this invention made under a government contract? If so, the agency and contract number:

NO

Description of Invention: *Please preserve all records of the invention and attach additional pages for the following. Each additional page should be signed and dated by the inventor(s) and witness(es).*

A. Description of the construction and operation of the invention (include appropriate schematic, block, & timing diagrams; drawings; samples; grap flowcharts; computer listings; test results; etc.)
B. Advantages of the invention over what has been done before.
C. Problems solved by the invention.
D. Prior solutions and their disadvantages (if available, attach copies of product literature, technical articles, patents, etc.).

Write in Dark Ink on Front Side Only, Please

Signature of Inventor(s): Pursuant to my (our) employment agreement, I (we) submit this disclosure on this date: [AUG. 4, 2003].

Employee No.	Name	Signature	Telnet	Mailstop	Entity & Lab Name
515623	AMIR SAID		857-3511	3U-3	ISL, HP LABS

(If more than four inventors, include additional information on another copy of this form and attach to this document)

Write in Dark Ink on Front Side Only, Please

INVENTION DISCLOSURE COMPANY CONFIDENTIAL PAGE ____ OF ____

Signature of Witness(es): *(Please try to obtain the signature of the person(s) to whom invention was first disclosed.)*

The Invention was first explained to, and understood by, me (us) on this date: [ ]

Full Name	Signature	Date of Signature
Debargha Mukherjee	Debargha Mukherjee	04/08/00
Nelson L. Chang	Nelson L. Chang	8/4/2000

Inventor & Home Address Information: *(If more than four inventors, include addl. information on a copy of this form & attach to this document)*

Inventor's Full Name: AMIR SAID

Street: 20380 STEVENS CREEK BLVD., APT. 208

City: CUPERTINO State: CA Zip: 95014

Do you have a Residential P.O. Address? P.O. BOX City State Zip

Greeted as *(nickname, middle name, etc.)* Citizenship: BRAZIL

Inventor's Full Name

Street

City State Zip

Do you have a Residential P.O. Address? P.O. BOX City State Zip

Greeted as *(nickname, middle name, etc.)* Citizenship

Inventor's Full Name

Street

City State Zip

Do you have a Residential P.O. Address? P.O. BOX City State Zip

Greeted as *(nickname, middle name, etc.)* Citizenship

Inventor's Full Name

Street

City State Zip

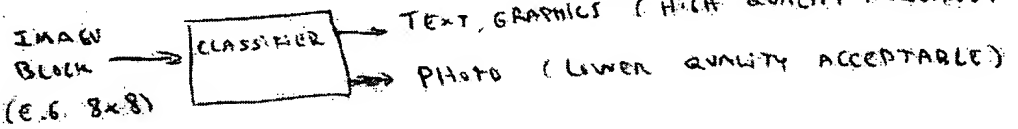
Do you have a Residential P.O. Address? P.O. BOX City State Zip

Write in Dark Ink on Front Side Only, Please

Description of Invention: *Please preserve all records of the invention and attach additional pages for the following. Each additional page should be signed and dated by the inventor(s) and witness(es).*

A. Description of the construction and operation of the invention (include appropriate schematic, block, & timing diagrams; drawings; samples; graphs; flowcharts; computer listings; test results; etc.)

THIS INVENTION PROVIDES A METHOD TO EFFICIENTLY AND QUICKLY IDENTIFY (CLASSIFY) IMAGE BLOCKS ACCORDING TO THEIR CONTENT AS SHOWN BELOW. (DETAILS ON REPORT HPL 1999-114)



B. Advantages of the invention over what has been done before.

THE INVENTION IS MUCH SIMPLER THAN METHOD THAT USE MORPHOLOGICAL ANALYSIS. IT USES ONLY DIFFERENCES BETWEEN PIXELS TO COMPUTE A HISTOGRAM. THE ENTROPY OF THE HISTOGRAM IS COMPUTED VERY EFFICIENTLY USING ONLY TABLE-LOOK-UP.

C. Problems solved by the invention.

DURING COMPRESSION OF COMPOUND DOCUMENTS IT IS ESSENTIAL TO IDENTIFY REGIONS WITH TEXT AND GRAPHICS, AND THOSE WITH PHOTOS. BECAUSE THEY ACCEPT QUITE DIFFERENT LEVELS OF IMAGE DEGRADATION.

THIS INVENTION PROVIDES A FAST, ONE-PASS, EFFECTIVE SOLUTION TO THE PROBLEM.

D. Prior solutions and their disadvantages (if available, attach copies of product literature, technical articles, patents, etc.).

- NEURAL NETWORKS: COMPUTATIONALLY MORE COMPLEX, REQUIRES TRAINING, NON-LINEAR OPERATIONS, ETC.
- CART (CLASSIFICATION OR REGRESSION TREES): REQUIRE COMPUTATION OF MANY PARAMETERS, COMPUTATIONALLY COMPLEX, NEEDS TRAINING.

Classification Methods for Compound Document Segmentation and Compression

Amir Said
Computer Peripherals Laboratory
HP Laboratories Palo Alto
HPL-1999-114
September, 1999

E-mail: said@hpl.hp.com

document analysis, page segmentation, image compression, text identification

A compound document, which may contain text, photos, and graphics in the same page, is better compressed if it is first segmented. The segmentation should identify the image components by type, and then apply the most suited image compression method to each type. The main problem when using this approach on hard copy devices is the computational complexity of the page analysis and segmentation. Those devices must compress and decompress high-resolution compound images in real time, and with limited resources. Thus, they need segmentation techniques with very low complexity. At the same time, the classification accuracy has to be good enough only for the compression purposes. We present a set of techniques to classify blocks of the compound page, following a one pass-approach. They are based on color analysis, edge-presence, and potential compression efficiency. We analyze the implementation details, and investigate how to keep the complexity at a minimum. The techniques are presented with their C++ implementation, and with images which exemplify classification results.

Classification Methods for Compound Document Segmentation and Compression

Amir Said
Hewlett Packard Laboratories
Palo Alto, CA

1 Introduction

Printers and scanners are required to process *compound documents*, which are composed of different types of visual information, like text (on different sizes, colors and fonts), color graphics (including color gradients, shading, etc.), and photos. With the increasing resolutions of the hard copy devices, large amounts of memory are necessary to store each document page after it has been rasterized. Digital image compression is essential to reduce the memory and bandwidth requirements, and consequently reduce the cost of the hardware.

The are two main problems with the compression of compound document images:

1. Different image types have different requirements for quality.
2. No single compression method is effective for all image types.

A solution to this problem is to pre-process the document, separating the areas according to the type of information they contain, and then compressing each area accordingly. For instance, we can separate the areas containing text, line-art, graphics, photos, color background, color gradients, etc. This is the classical *page decomposition* problem, which is also used for applications other than compression. For instance, it can be used for optical character recognition (OCR) [7], document analysis [8], indexing, etc. However, in this document, we are concerned only with its use for compression.

There is variety of techniques developed for page decomposition, and there is active research on solutions adapted to compression needs [1,3,5]. Unfortunately, the most sophisticated page decomposition algorithms have very high complexity, which makes them useless for embedded solutions on hard copy hardware.

Our objective is to develop segmentation techniques that yield the most improvement in compression, and in the quality of the compression documents, using a minimal amount of computational resources beyond those required for compression. In the next section we describe the factors used to measure complexity, and how they orient the algorithm design process.

Our proposed techniques are classified as:

- *Color analysis*. One of the most important objectives of page analysis is to find the regions with text and graphics, which need special compression. Since in typical documents those regions have only a few distinct colors, we try to identify those regions with color analysis. We show that the problem can be solved in a very efficient manner, with only a few comparisons per pixel.
- *Color identification*. Frequently a large part of a document page corresponds to one or two predominant colors. For example, most document pages have a white background. Compression methods can be more efficient with *a priori* knowledge that a page has such predominant color. We propose a method to quickly identify if predominant colors are present. Surprisingly, the test can be performed with a high degree of confidence by testing only a small number of pixels.
- *Edge occurrence*. This classification method adds a "safety net" for the cases when the color analysis cannot identify the regions of text and graphics (e.g., when the document page is blurred). It identifies the presence of the edges typically found in regions with text and graphics, but it does not locate the edges in the block. Thus, it is less complex than techniques that explicitly look for edge location.
- *Compression efficiency*. Each compression methods is most efficient (in a relative manner) for particular image type. We propose a parallel hardware implementation that can exploit this fact to do simultaneous compression and classification. There is little overhead in the classification and practically no delay between classification and compression because the class is identified directly from the compressed data.

2 Design Objectives

Computational complexity is the main issue for use of page decomposition in hard copy devices. The required high throughput—in pages per minute—mandates the use of specialized hardware or very fast firmware.

To minimize costs, the segmentation method is required to have the following properties.

1. *Small buffers*: the decision must be based on data contained in a small amount of memory.
2. *Low bandwidth*: multiple passes on image regions should be avoided.
3. *Fast computation*: the classification decision should be obtained with a small amount of operations.

The techniques presented in this document address all these issues simultaneously. They are meant to produce a page decomposition that is good enough to improve compression, while minimizing the complexity to a level compatible with real-time use on hard-copy devices.

1. The memory problem is solved by making decisions on small image blocks (e.g., 8×8 blocks).
2. The bandwidth problem is solved by processing a block only once before deciding on its type.
3. The computation problem is solved by minimizing the number of operations and comparisons, or by using data that has already being computed by the compression method.

The computational complexity for classification can be minimized in two ways:

(a) Use fast algorithms to compute the data required for classification.

(b) Use data that has to be computed for compression anyway (e.g., discrete cosine transform coefficients).

The first three techniques proposed in this document are based on (a). They represent very significant reductions in complexity, compared to previously published techniques. For instance, the technique proposed in [1] requires the computation of 11 different numbers, each requiring several operations (including multiplication) per pixel. Furthermore, those numbers are sequentially tested using a decision tree, which can be expensive on CPU with long pipelines. Our proposed techniques use only a few comparisons, and possibly, additions or bit shifts per pixel. As an additional advantage, the classification can help eliminating some computation-intensive tasks, like computation of the Discrete Cosine Transform (DCT).

The fourth proposed technique is based on complexity-reduction technique (b), which is truly useful only when the computations really cannot be avoided, or when there is no gain avoiding it (normally in parallel hardware structures). Several techniques have been proposed to use transformed data [1,2,3,5], for classification. The problem is that the best transforms for compression may not be the best for classification. For instance, the DCT destroys edge information. Our proposed method consists of a parallel implementation of both transforms and coding. It exploits the fact that only the combination of an image transforms, *plus* the matched coding algorithms, can effectively discriminate block types. As an additional benefit, this scheme uses the data that will form the compressed stream for classification, eliminating the delay between classification and compression.

3 Fast Classification Techniques

Figure 1 shows how the classification method described in this document is used in a system for compound document compression. First, the image is divided in blocks. Each block is analyzed by the block classification function, which puts out its decision on the block class. Next, the block compression function will encode the image data, using the classification to select the most suited compression method. We assume that the block class is efficiently included in the compressed bit stream, so that the decoder may know which decompression method to use.

The output of the classification function depends on the classification technique. For instance, it may classify the block as part of a document area containing: (a) text, (b) graphics, and (c) photo. Other class names may be less intuitive, but more suited for the block compression function. For example, blocks can be classified as (a) smooth, (b) filled with edges, and (c) noisy, giving a better indication of the best compression method for each case.

For completeness, we assume that classification function may use some side information. This can be the results of the classification on previous blocks, or any other information that may be improve the accuracy of the classification.

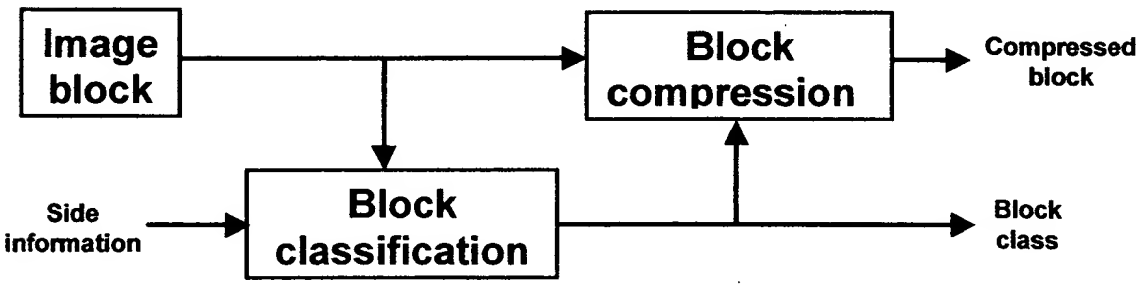


Figure 1 - Scheme for sequential block classification and compression.

Below we describe techniques that can be used independently, or that can combined for improved classification. To simplify the presentations, the techniques are put together according to the type of information they use.

3.1 Classification Using Color Information

We assume that each image pixel is represented by a vector (R, G, B), where R, G, and B, are the red, green, and blue components, respectively. We say that two pixels have the same color when their (R, G, B) components are identical. We use the RGB color space for convenience: the techniques can be used for other color spaces, like YUV, YCbCr, LAB, etc.

One fundamental difference between areas containing text and graphics, and areas containing photos, is that the first contains only a small number of distinct colors, while in the latter, typically each pixel has a slightly different color. When the resolution is high enough, even features like color gradients in the background satisfy this condition (because the color changes slowly, and normally less than two or three colors are present in each small block).

Let N_c be the number of distinct colors in a given block, and let M_c be a (small) positive number. We can classify the block as:

(a) *Discrete color*: if $N_c \leq M_c$.
(b) *Continuous color*: if $N_c > M_c$.

Box A shows how a compression algorithm can use this form of classification.

Box A - Block compression algorithm based on color analysis.

1. Find N_c, the number of distinct colors in a block.
2. If $N_c > M_c$
   (a) encode block class as "0",
   (b) encode block pixels using a technique for continuous tone images.
3. If $N_c \leq M_c$
   (a) encode block class as "1",
   (b) encode N_c,
   (c) encode the N_c distinct colors, assigning indexes 0, 1, 2, 3, ... to each,
   (d) if $N_c > 1$ then encode the color index of each block pixel.

The main advantages of this technique are

- A large fraction of a typical document page can be classified as discrete-colors. Finding these areas corresponds to solving a great part of the page analysis problem.
- If M_c is small, then the classification has very small complexity.
- On discrete-color blocks, even extremely simple coding methods can give some reasonably good compression.

Table I shows the number of bytes used by simply representing the (R, G, B) vectors with 24 bits, and the color index with the smallest number of bits required to represent the N_c color indexes. Note that the compression ratios are remarkably good, considering the how crude this

coding method is. They show that when a small number of colors is present, even in the worst case we can get good compression. Furthermore, it is possible to obtain significantly better compression using more sophisticated coding of the color indexes.

Table I - Compression ratios (CR) obtained by coding 8×8 blocks, classified as discrete-color type, using a binary representation for the color vectors and color indexes.

N_c	1	2	3	4	5	6	7	8	10	12	15
bytes	3	14	25	28	39	42	45	56	62	68	77
CR	64.0:1	13.7:1	7.7:1	6.9:1	4.9:1	4.6:1	4.3:1	3.4:1	3.1:1	2.8:1	2.5:1

Box B shows a C++ function that detects the number of distinct colors in a block, and returns zero if the number is larger than the predefined maximum. We assume that each pixel (R, G, B) vector is stored in a single 32-bit integer. Note that the relative complexity (per block pixel) is proportional to maximum number of colors. If $M_c = 2$, then the complexity is roughly two comparisons per pixel, which is near the practical lower bound for any classification method.

Box B - C++ function for the determination of the number of distinct colors in a block.

```
int Number_of_Colors(int max_colors,
                     int number_of_pixels,
                     int rgb_vector[],
                     int block_colors[])
{
  int c, distinct_colors = 1;
  block_colors[0] = rgb_vector[0];
  for (int n = 1; n < number_of_pixels; n++) {
    int v = rgb_vector[n];
    for (c = 0; c < distinct_colors; c++)
      if (block_colors[c] == v)
        break;
    if (c == distinct_colors) {
      if (distinct_colors == max_colors)
        return 0;
      block_colors[distinct_colors++] = v;
    }
  }
  return distinct_colors;
}
```

One problem with the function of Box B is that it does not work well with noisy scanned images. It is necessary to generalize this technique accept a certain tolerance for each color component. This is done by keeping track of maximum and minimum pixel values for individual color components. The function that works with all the color components simultaneously is somewhat long, so we present only a simpler version. Box C shows a function for finding the number of

grayscale levels in a block. Note how it keeps track of the tolerance by updating a [minimum, maximum] interval. The function for RGB pixels is similar, with vectors replacing the maximum and minimum interval.

The function of block C can be applied to each color component separately, and each can be coded independently. However, it is important to note that color transitions normally happen in all color components simultaneously, so it would be less efficient to code them independently.

Box C - C++ function for the determination of the number of distinct grayscale levels in a block, under an error tolerance.

```
int Number_of_Levels(int max_levels,
                     int tolerance,
                     int number_of_pixels,
                     int color_component[],
                     int minimum[],
                     int maximum[])
{
  int c, distinct_colors = 1;
  minimum[0] = maximum[0] = color_component[0];
  for (int n = 1; n < number_of_pixels; n++) {
    int g = color_component[n];
    for (c = 0; c < distinct_colors; c++)
      if (p < maximum[c]) {
        if (maximum[c] - g < tolerance) {
          if (p < minimum[c]) minimum[c] = g;
          break;
        }
      }
      else {
        if (g - minimum[c] < tolerance) {
          if (g > maximum[c]) maximum[c] = g;
          break;
        }
      }
    if (c == distinct_colors) {
      if (distinct_colors == max_colors)
        return 0;
      minimum[distinct_colors]   = g;
      maximum[distinct_colors++] = g;
    }
  }
  return distinct_colors;
}
```

It is clear that the function in Box C requires more operations and comparisons than the function in Box B. It is possible to avoid such increase in complexity by using an approximation. We can disregard the least significant bits of the color values, so that small changes may not affect the outcome. Box D shows how this can be done using a binary mask. Note how similar it is to the function in Box B, having practically the same complexity. However, this function does not

guarantee that all blocks are classified correctly, it may return a number of colors larger than that returned by the function of Box C. However, the effect in compression ratio may be small.

Box D - C++ function for the determination of the number of distinct colors in a block, with error tolerance supported by binary mask.

```
int Number_of_Colors(int max_colors,
                     int number_of_pixels,
                     int rgb_vector[],
                     int block_colors[])
{
  const int Mask = 0xFCFCFC;
  int c, distinct_colors = 1;
  block_colors[0] = rgb_vector[0] & mask;
  for (int n = 1; n < number_of_pixels; n++) {
    int v = rgb_vector[n] & Mask;
    for (c = 0; c < distinct_colors; c++)
      if (block_colors[c] == v)
        break;
    if (c == distinct_colors) {
      if (distinct_colors == max_colors)
       return 0;
      block_colors[distinct_colors++] = v;
    }
  }
  return distinct_colors;
}
```

In all the functions we have shown, the number of comparisons per pixel is proportional to the number of distinct colors in a block (up to the maximum allowed). Thus, these functions are not efficient when it is desired to identify blocks with many different colors. In those cases, it again possible to use an approximation that works in most cases, and greatly reduces the complexity.

The technique is based on splitting the color space in different "bins," and testing the colors in each bin. It is based on the fact that blocks with graphics normally have quite distinct colors. Photos, and random patterns, on the other hand, have a large number of slightly different colors.

Box E shows an implementation of this technique, with the color space divided in 8 bins. This choice of the number of bins is meant to minimize complexity: the bin number can be computed with simple binary operations on the color components. In this version the maximum number of comparisons per pixel is proportional to the maximum number of colors per bin, and not the maximum number of colors per block. To make the function smaller, we also assume that the maximum number of distinct colors is 32.

A disadvantage of this method is that the information about bins has to be initialized every time the function is called. Thus, it is better to keep the number of bins small, and not use this function when the maximum number of colors is small.

Box E - Function for the determination of the number of distinct colors in a block, with complexity independent of the total number of colors.

```
int Number_of_Colors(int max_colors,
                     int max_colors_per_bin,
                     int number_of_pixels,
                     int rgb_vector[],
                     int block_colors[32])
{
  int c, distinct_colors = 0;
  int bin_colors[8][32];
  int bin_distinct_colors[8] = { 0, 0, 0, 0, 0, 0, 0, 0 };
  for (int n = 0; n < number_of_pixels; n++) {
    int v = rgb_vector[n];
    int bin = ((v >> 7) & 1) | ((v >> 14) & 2) | ((v >> 21) & 4);
    for (c = bin_colors[bin] - 1; c >= 0; c--)
      if (bin_colors[bin][c] == v)
        break;
    if (c < 0) {
      if (distinct_colors == max_colors)
        return 0;
      if (bin_colors[bin] == max_colors_per_bin)
        return 0;
      block_colors[distinct_colors++] = v;
      bin_distinct_colors[bin_colors[bin]++] = v;
    }
  }
  return distinct_colors;
}
```

The application of these classification algorithms to compound images shows how effective color classification can be for compression of high-resolution images. Figures 2 and 4 show parts of a 600 DPI image. Figures 3 and 5 show color maps of the respective classification results, using the function on Block D. An amber tone is used for 8×8 blocks with at most two colors, and a blue tone for the other blocks.

Note that although a significant part of Figure 2 has color gradients, the pixel values change slowly enough so that most blocks are classified as having only two colors. This is very important for compression. Figure 5 shows how the color classification is useful for separating photos from graphics and text. Some blocks inside the photo do have changes that are small enough for classification as discrete-colors, but this can be easily identified via a local analysis.

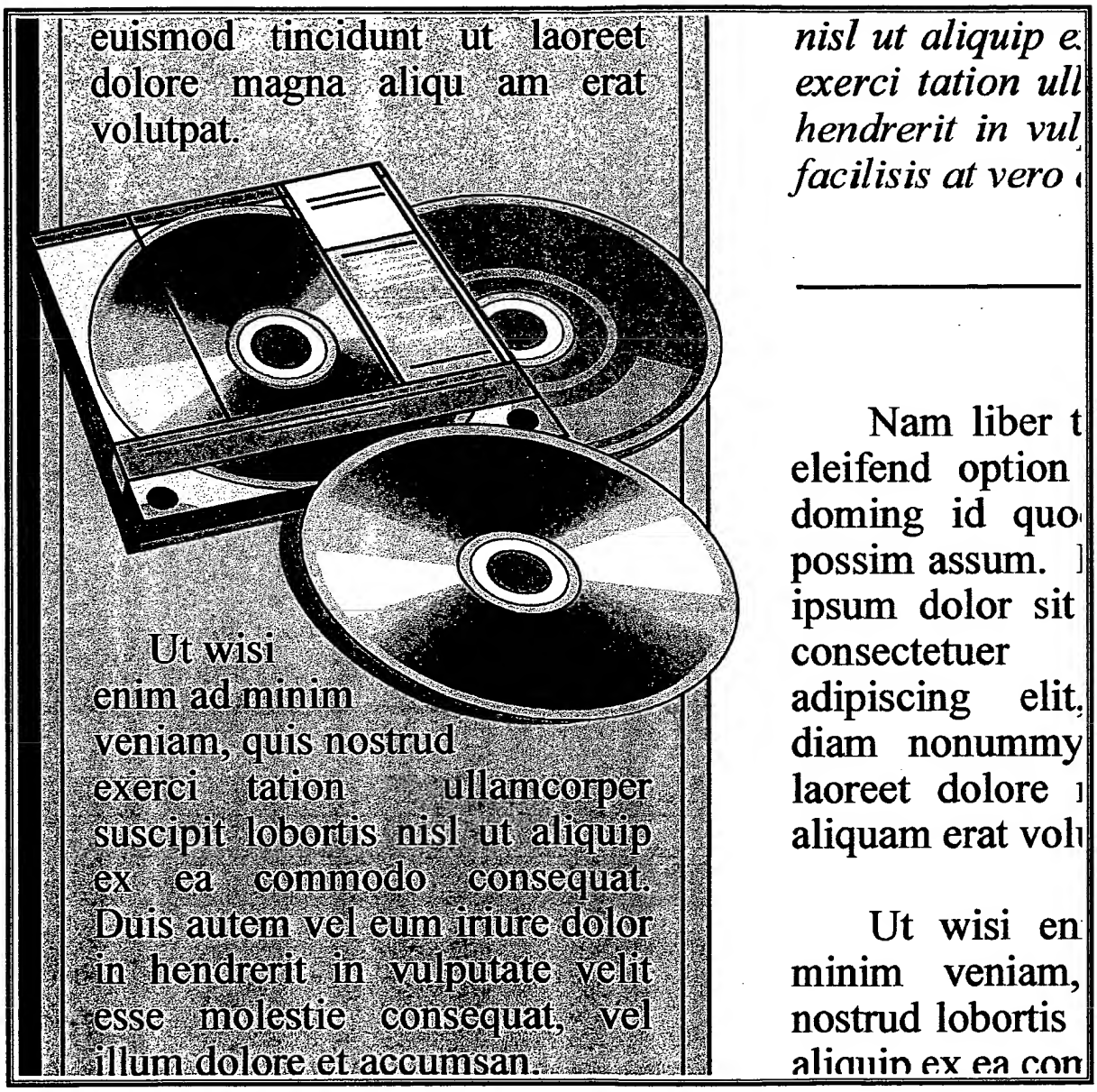


Figure 2 - Image "CDs" used for testing the classification methods.

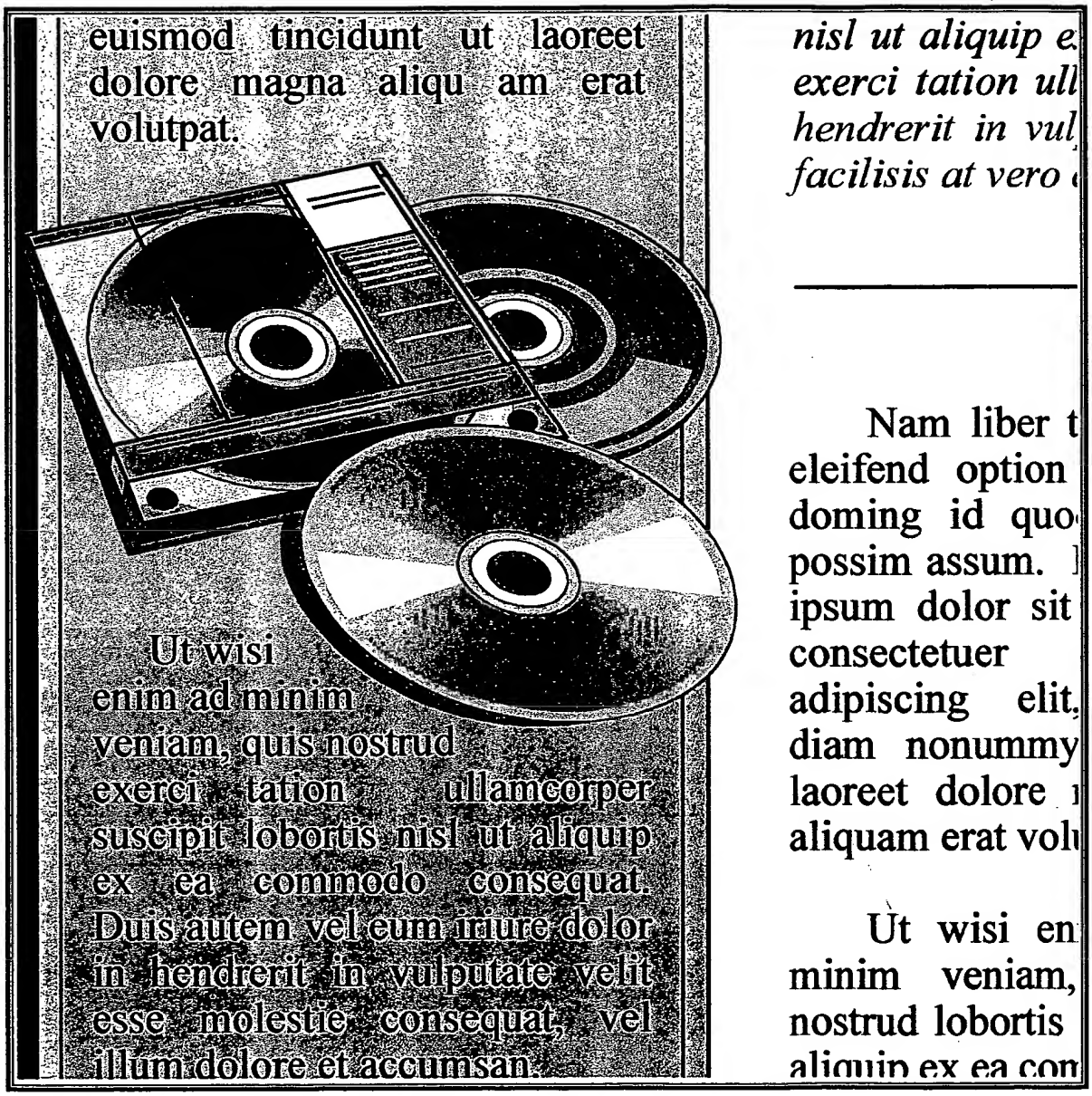


Figure 3 - Color-based classification implemented on 8×8 blocks of image "CDs." The classification results are color mapped: amber regions have at most two distinct colors (within ±2 tolerance), and blue regions have more than two colors.

Figure 4 - Image "Faces" used for testing the classification methods.

Figure 5 - Color-based classification implemented on 8×8 blocks of image "Faces." The classification results are color mapped: amber regions have at most two distinct colors (within ±2 tolerance), and blue regions have more than two colors.

3.2 Predominant Color Identification

Compound documents frequently have one or more predominant colors. For example, many documents have a predominant white background. The knowledge of those predominant colors can be advantageous in several circumstances. One of them is when coding the data obtained with the color-based classification scheme defined in the previous section. For instance, when a block is identified as having only a few distinct colors, the block can be more efficiently coded if it is known *a priori* which are the most common.

There are two ways to identify the most common colors: data about colors can be gathered during the one pass classification; or we can have a preliminary pass on the image just to identify the predominant colors. The first case is equivalent to adaptive coding, and can use any of the techniques developed for that form of coding. Its main limitation is that it is strictly causal, and can be misled by local properties in the beginning of the adaptation process. In fact, this problem is the main motivation for a two-pass scheme. It contradicts our assumption that we use only one pass for classification, but we show that it is possible to identify the colors by sampling only a very small fraction of the pixels. In short, the preprocessing complexity (including bandwidth) of predominant color identification is far smaller than that of a full pass.

We next show that how to use statistical analysis to identify predominant colors by testing only a relatively small number of pixels, and how to simultaneously minimize the complexity of the test process.

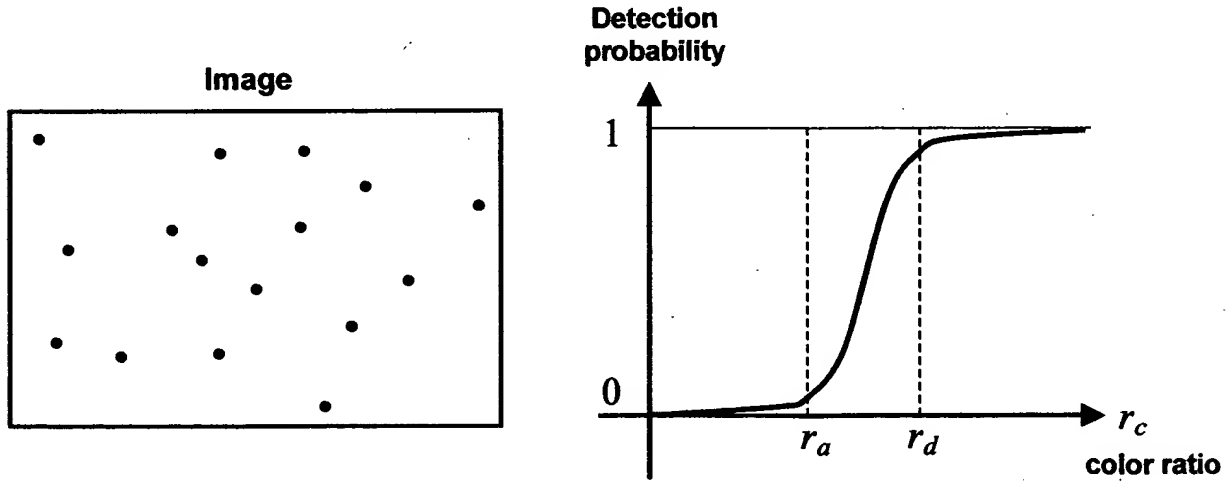


Figure 6 - Assumptions used by identification algorithm: (a) pixels are sampled from random image locations; (b) detection rules are based on an acceptable ratio r_a and a desirable ratio r_d.

Let r_c be a color ratio, i.e., the number of pixels with the color c divided by the total number of pixels in the image. We want to identify, with minimal complexity, all colors that have ratio r_c sufficiently large. In other words, we want to create a detection rule that has positive outcome if r_c is expected to be sufficiently large, and negative otherwise.

Figure 6 shows the assumptions used by the identification algorithm. First, we assume that the pixel (R, G, B) vectors are tested sequentially following a random pattern. This assumption guarantees that the pixel samples are statistically independent, and that we can use standard statistical tests.

Next, we define two parameters: an acceptable ratio r_a and a desirable ratio r_d. The objective is to have a detection rule that minimizes the probability of two events:

(a) *False-positive outcome*: a color with a true ratio r_c smaller than r_a is identified as a predominant color.

(b) *False-negative outcome*: a color with a true ratio r_c larger than r_d is not identified as a predominant color.

As shown in Figure 6, the difference between r_a and r_d allows for a smooth transition in the detection probability. The slope of the transition depends on the number of tests, as we cannot have a steep slope with a small number of tests.

The color detection algorithm uses a color occurrence list, L, of records in the form [(R,G,B), q], where (R,G,B) is a color vector, and q is the number of times pixels with that color vector had been selected. Let P_n, T_n, and U_n, $n = 1, 2, \ldots, N$, be sequences of non-negative numbers, such that $T_1 = 1$, and $T_n < T_{n+1}$ .The detection algorithm is shown on Box F.

Box F - Algorithm for the detection of predominant colors.

1. Reset the color occurrence list L (set as an empty list).
2. Test P_1 pixels. For each pixel, if its (R,G,B) vector is in the color occurrence list L, then increment its counter q by one. Otherwise add the (R,G,B) vector as a new entry of the list L, and set its counter q to one.
3. Repeat for $n = 2, 3, 4, \ldots, N$.
   (a) Test P_n pixels. For each pixel, if its (R,G,B) vector is in the color occurrence array L, then increment its counter q by one.
   (b) Remove from list L all entries with counter $q < T_n$.
   (c) If list L is empty or all entries have counter $q > U_n$ then go to step 4.
4. Output the entries in L as the predominant colors and stop.

Since each pixel color has to be compared with all vectors in the list L, it is interesting to minimize the number of elements in L, and this is done by eliminating the less common colors.

The number of pixel tests, P_n, and the corresponding thresholds, T_n and U_n, have to be carefully chosen so that no predominant color is removed in this process.

In order to compute the optimal set of parameters P_n, T_n, and U_n, we define the probability functions

$$f(l,k,\rho) = \binom{k}{l} \rho^l (1-\rho)^{k-l} = \frac{k!\,\rho^l (1-\rho)^{k-l}}{l!(k-l)!},$$

$$v(l,k,\rho) = \sum_{i=0}^{l} f(i,k,\rho).$$

The probability that a color with ratio $r_c \geq r_d$ is discarded at step 2, F_{FP}^1 (false-positive probability), is upper bounded by

$$F_{\text{FP}}^1 \leq f(0, P_1, r_d).$$

The probability that a color with ratio $r_c \geq r_d$ is discarded at the n-th step 3(c), F_{FP}^n, is determined not only by the number of occurrences, but also considering if it was not discarded in a previous step. The exact formulas are somewhat complex, and need to be computed in a recursive manner. They are presented later. For now, we use approximations, ignoring the possibility of early removal from the list.

First, we define

$$S_n = \sum_{m=1}^{n} P_m,$$

and then use it in the approximation for false-negative classification

$$F_{\text{FN}}^n \leq v(T_n - 1, S_n, r_d).$$

Similarly, we can estimate an upper bound on the probability of a false-positive decision on step 3(d) using

$$F_{\text{FP}}^n \leq 1 - v(U_n - 1, S_n, r_a).$$

We can use the equations above to select set the parameters P_n, T_n and U_n, and have a sufficiently small probability of incorrect decisions. At step 4, when the algorithm stops, the false-positive probability is bounded as

$$F_{\text{FP}} \leq 1 - v(T_N - 1, S_N, r_a),$$

and the false-negative probability is bounded as

$$F_{\text{FN}} = F_{\text{FN}}^N \leq v(T_N - 1, S_N, r_d).$$

Table II shows an optimal strategy computed for the set of parameters listed in its caption. It also shows the value T_N, which is the threshold optimized under the assumption that the algorithm must stop at that step n. Note how the predominant colors can be identified precisely testing only

up to 120 pixels, independently of the image size. Later we show that in practice a much smaller number is tested, because colors can quickly satisfy the early detection criteria, and the worst case is quite rare.

Table II - Optimal sequence of parameters computed with approximate probabilities and under the following test conditions: acceptable ratio $r_a = 0.2$, desired ratio $r_d = 0.4$, early decision false-negative and false-positive probabilities $F_{FN}^n \leq 0.001$, $F_{FP}^n \leq 0.001$, final decision false-negative and false-positive probabilities $F_{FN} \leq 0.01$, $F_{FP} \leq 0.01$. The values of F_{FN} and F_{FP} are based on the optimal final threshold T_N.

n	T_n	T_N	P_n	S_n	U_n	F_{FN}	F_{FP}
1	1	5	14	14	9	0.2792	0.1298
2	2	6	5	19	11	0.1629	0.1630
3	3	8	5	24	12	0.1919	0.0892
4	4	9	4	28	14	0.1485	0.0900
5	6	11	7	35	16	0.1122	0.0747
6	8	13	8	43	18	0.0695	0.0733
7	10	15	7	50	20	0.0540	0.0607
8	12	17	6	56	22	0.0517	0.0432
9	15	20	10	66	25	0.0395	0.0306
10	18	23	10	76	28	0.0304	0.0219
11	21	26	10	86	30	0.0235	0.0157
12	23	28	9	95	33	0.0127	0.0178
13	28	32	12	107	36	0.0118	0.0095
14	36	36	13	120	39	0.0090	0.0059

For an exact computation of probabilities, we define $B_n(k, r_c)$ as the probability of a given color with ratio r_c occurring i times and not being discarded up to step $n > 1$. The probability that this color is discarded at that step is

$$B_d^n(r_c) = \sum_{k=T_{n-1}}^{T_n-1} \sum_{l=T_{n-1}}^{k} B_{n-1}(l, r_c) f(k-l, P_n, r_c)$$

After using this equation to compute P_n and T_n we update probabilities using the following recurrence

$$B_n(k, r_c) = \begin{cases} \sum_{l=T_{n-1}}^{k} B_{n-1}(l, r_c) f(k-l, P_n, r_c), & T_n \leq k \leq S_n \\ 0, & 0 \leq k < T_n \end{cases}$$

The probabilities of incorrect decision at step n are

$$F_{\mathrm{FN}}^{n} \leq B_{d}^{n}(r_{d}),$$

and

$$F_{\mathrm{FP}}^{n} \leq \sum_{k=U_{n}}^{S_{n}} B_{n}(k, r_{a}).$$

The probabilities of false-positive and false-negative decisions when the algorithm stops at step 4 are bounded, respectively, by

$$F_{\mathrm{FP}} \leq 1 - B_{d}^{N}(r_{a}),$$

and

$$F_{\mathrm{FN}} \leq B_{d}^{N}(r_{d}).$$

Table III shows the optimal test sequences computed with these exact probability computations. Note that the required number of tests is smaller than those in Table II, which means that the approximations are easy to compute, but are too conservative.

As the final consideration for minimizing the complexity of this algorithm, we need an efficient way to implement the list of color entries L. When r_a is relatively large (e.g., 0.3) only a very small number of entries can possibly be in L, and no special implementation may be necessary. For smaller ratios, we may need implementations that are more sophisticated, like:

(a) Keep L as a sorted list [6], so that pixels are compared first to the most common colors.
(b) Implement L as a hash table [6] in order to minimize the number of color comparisons.

We applied the algorithm of Box F in the images of Figure 7 and 8. We used a test schedule determined by the exact probability formulas, and implemented L as a sorted list. The search parameters are: acceptable ratio $r_a = 0.1$, desired ratio $r_d = 0.3$, early decision false-negative and false-positive probabilities $F_{\mathrm{FN}}^{n} \leq 0.001$, $F_{\mathrm{FP}}^{n} \leq 0.001$, final decision false-negative and false-positive probabilities $F_{\mathrm{FN}} \leq 0.01$, $F_{\mathrm{FP}} < 0.01$. These were chosen as typical required for compound document compression—colors with smaller ratios may not be very useful for compression purposes.

Table III - Optimal sequence of parameters computed with exact probabilities and under the following test conditions: acceptable ratio $r_a = 0.2$, desired ratio $r_d = 0.4$, early decision false-negative and false-positive probabilities $F_{FN}^n \leq 0.001$, $F_{FP}^n \leq 0.001$, final decision false-negative and false-positive probabilities $F_{FN} \leq 0.01$, $F_{FP} \leq 0.01$. The values of F_{FN} and F_{FP} are based on the optimal final threshold T_N.

n	T_n	T_N	P_n	S_n	U_n	F_{FN}	F_{FP}
1	1	5	14	14	9	0.2792	0.1298
2	2	6	4	18	10	0.2006	0.1329
3	3	7	3	21	11	0.1882	0.1085
4	4	8	4	25	13	0.1374	0.1088
5	6	10	7	32	15	0.0852	0.0880
6	8	12	5	37	17	0.0960	0.0503
7	10	14	6	43	18	0.0785	0.0346
8	12	15	6	49	20	0.0267	0.0447
9	15	18	9	58	23	0.0142	0.0229
10	18	21	7	65	25	0.0201	0.0100
11	21	24	8	73	27	0.0162	0.0047
12	26	26	8	81	29	0.0036	0.0034

Figure 7 - Image "Dinosaur Head" (940×400) used for testing the predominant color detection algorithm. Colors white and green were detected after testing 80 pixels, 134 RGB comparisons.

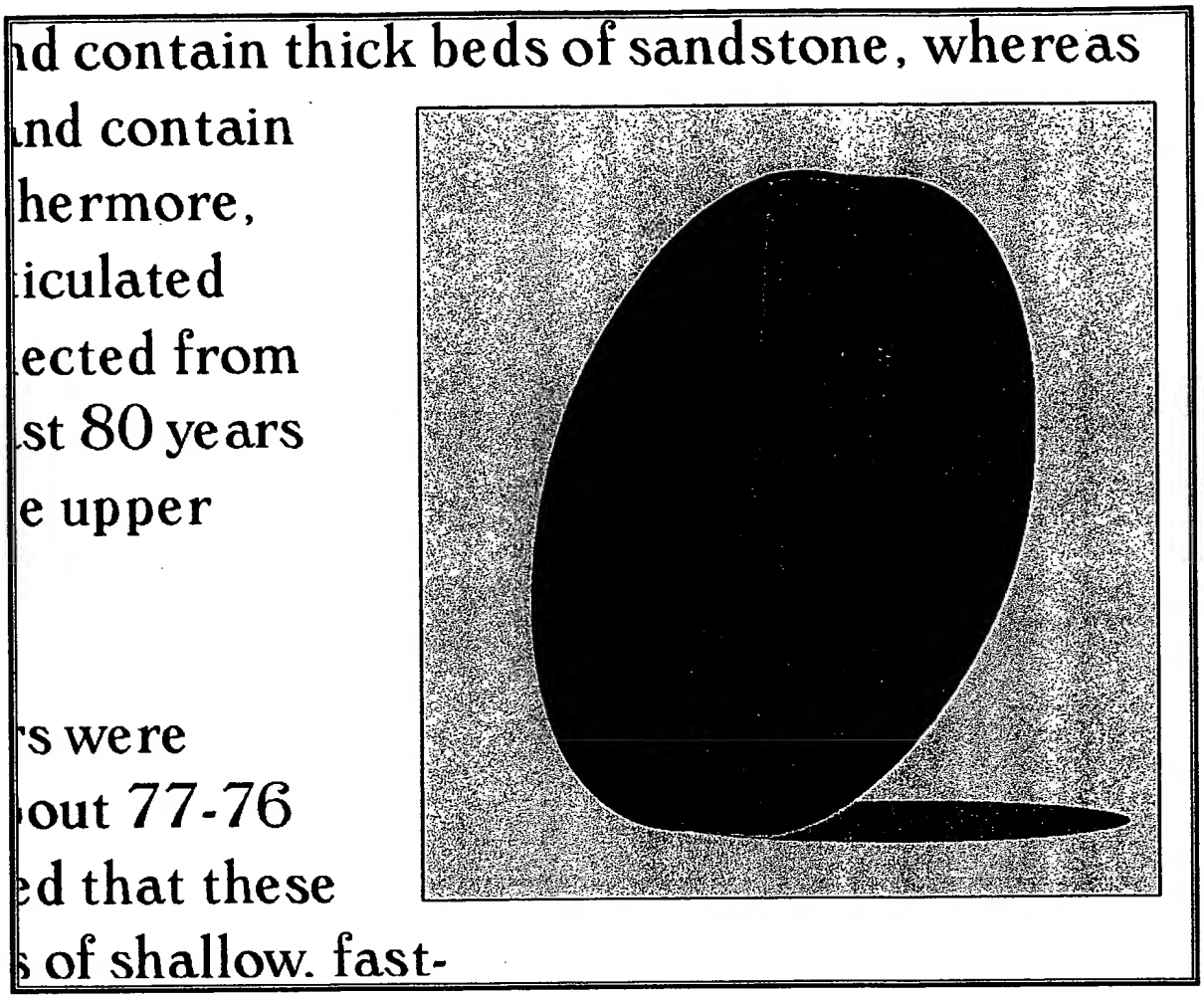


Figure 8 - Image "Monthly Graphics" (840×700) for testing the predominant color detection algorithm. Colors white and gray were detected after testing 54 pixels, 129 RGB comparisons.

The predominant colors in the image of Figure 7, white and green, were detected after testing 80 pixels. The number of (R,G,B) vectors comparisons was 134. Note that this represents an average of less than two comparisons per tested pixel. The predominant colors in the image of Figure 8, white and gray, were detected after testing 54 pixels, using 129 comparisons.

When applied to natural images (photos) the algorithm quickly terminates with an empty list. When there is a photo, plus a predominant color, like in Figure 4, it may require more comparisons, but it is still a quite reasonable number. For example, the algorithm detects the color white in the image of Figure 4 after testing 67 pixels, and using 357 comparisons. On the image of Figure 2 white is detected after testing 45 pixels, 202 comparisons.

3.3 Classification Using Entropy of Distributions

One property of image areas containing text and graphics is that they contain jumps in luminance values corresponding to on the edges of letters or lines. This is very important for compression, because lossy methods based on linear transforms (e.g., discrete cosine transform, discrete wavelet transform) do not compress those edges efficiently. They require too many bits, and may produce very objectionable artifacts around text.

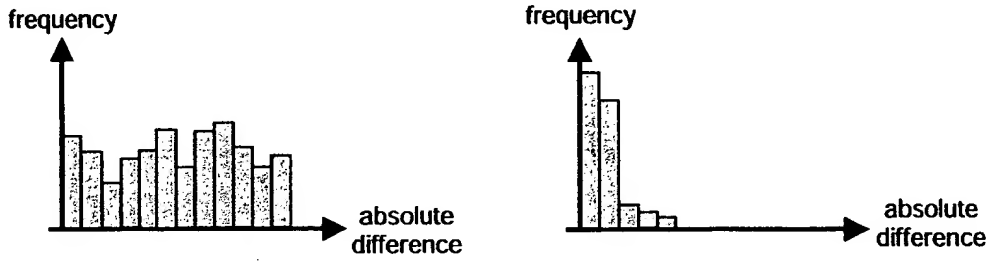
Since features like edges are unrelated to color representations, in this section we consider only a grayscale representation of the image pixels. In other words, the classification techniques presented next may be used on the gray or Y color component (color plane).

There are many methods for edge detection. However, for our purposes we do not need to know the location of the edges, we only need to know if they are present or not. We can get some information about the presence of edges in a block by computing, for each pixel $p_{i,j}$, the distribution of the horizontal and vertical absolute differences

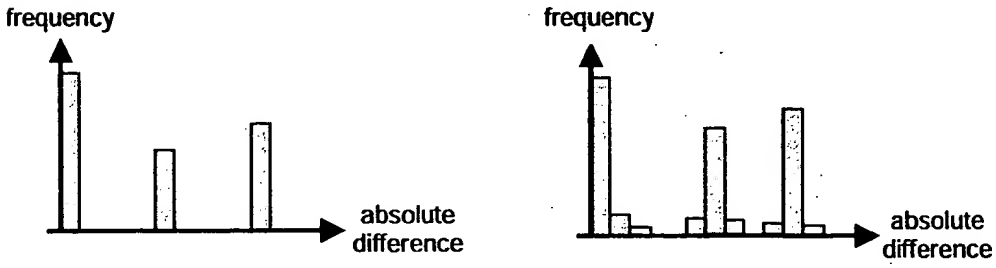
$$d^h_{i,j} = \left| p_{i,j} - p_{i,j-1} \right|$$

$$d^v_{i,j} = \left| p_{i,j} - p_{i-1,j} \right|$$

and then analyzing the histogram of these absolute differences.



(a) Natural images (photos), graphic arts, complex patterns.



(b) Text, graphics (possibly blurred).

Figure 9 - Typical histograms of absolute difference between adjacent pixels.

Figure 9 shows examples of typical histograms. Blocks that are part of photos or complex graphic patterns have histograms that are flat or are peaked at zero (flat areas on photos). Blocks with text and graphics normally have histograms containing a few isolated peaks. Sometimes the edges are blurred, producing peaks that are not isolated, plus some random differences.

We wan to use this property of the histograms to identify the blocks that have edges, but we also want to minimize the computational effort analyzing the histogram. The ideal is to have a single number that, when compared to an absolute threshold, indicates if the block has important edges or not. One function that possesses some of the desired is the entropy function.

Let h_n, $n = 0, \ldots, C$, be a set of non-negative numbers corresponding to a histogram, and let

$$T = \sum_{n=0}^{C} h_n$$

The entropy function is defined by

$$\psi(h) = \sum_{h_n \neq 0} \frac{h_n \log(T / h_n)}{T} = \log(T) - \frac{1}{T} \sum_{h_n \neq 0} h_n \log(h_n)$$

In addition, we define the maximum argument in the histogram as

$$\mu(h) = \max_{h_n \neq 0} \{n\}$$

An advantage of the entropy function is that its value does not depend on the exact location of the peaks on the histogram, but only on their shape. However, for our purposes a peak at value zero does have a special significance: it represents flat areas, which should be disregarded by the edge detection. We use the maximum difference to quickly know if there is only one peak at zero, or more a different distribution.

Table IV shows the expected magnitude of these two parameters in different image block types. We can see that on blocks that contain text we can expect the entropy to be low, and the maximum difference to be high.

Table IV - Properties of the entropy function and maximum difference.

Image block features	entropy	maximum difference
smooth block	low	low
large random differences	high	high
large differences due to edges	low	high

A function that can be used to identify this condition is

$$\Psi(h) = \frac{a(\psi(h) + b)}{\mu(h) + c}$$

where a, b, and c are constants that normalize the entropy, and change the sensitivity of the classification. They should be chosen so that $\Psi(h)$ is small when the image block contains edges, and $\Psi(h)$ is large in all other cases. Under those conditions, a block is classified as having edges (and thus need special care during compression) if $\Psi(h)$ is smaller than a given threshold, like $\Psi(h) < 1$.

Since $\Psi(h)$ is defined using logarithms and multiplication, it may appear to be computationally costly. However, it is quite the opposite. The function factors can be pre-computed, scaled and rounded to integers, so that all complex computations are replaced by table look-up. Box G shows a C++ function using those techniques. This function is meant to be used for blocks with 64 pixels, and we assume that the pixel values are scalars (not color vectors), in the range [0,255]. Note that the function uses only additions and comparisons (no multiplication).

Another way to allow a distinction between blocks with edges and smooth blocks is to use an expurgated version of the entropy. We can define a positive number ρ (e.g., $\alpha = 8$), and compute

$$T_e = \sum_{n=\alpha}^{C} h_n$$

$$\psi_e(h) = \sum_{n=\alpha, h_n \neq 0}^{C} \frac{h_n \log(T_e / h_n)}{T_e}$$

All the definitions above are bases on simple absolute difference between pixels, which are known to be directly related to edge-height [4]. A similar type of histogram property (isolated peaks) was found in some transformed images [2]. The algorithms above can be directly applied to such transformed images. However, each isolated peak in the histogram of differences is mapped to several peaks in the transform histogram, which reduces the discrimination capabilities of the method above.

Figures 10 and 11 show the result of applying the algorithm of Box G to images on Figure 2 and 4, respectively. Note how it has given a positive outcome at the edges of text, but not on its "interior" (the white or black space inside the letters). Similarly, it does not detect edges that occur in the boundary between blocks.

This is consistent with our objectives that classification will be used only for compression purposes. The color transition on the text edges have to be compressed in a special manner to avoid artifacts like blurring, ringing, checkerboard patterns, "mosquitoes," etc. The constant-color spaces inside the letters do not have discontinuities, and thus cannot create artifacts. We are also assuming a block-based compression, which will not be affected by edges in the boundaries.

Box G - Function to identify image blocks with edges using the entropy function and maximum difference.

```
int Block_Has_Edges(int pixel[])
{
  int diff, k, e = 0, max = 0;
  static int * histogram = calloc(256, sizeof(int));
  const int AnalysisTable[113] = { 0,
     3983,  6796,  9168, 11252, 13123, 14824, 16384, 17823,
    19156, 20395, 21549, 22627, 23634, 24576, 25458, 26283,
    27056, 27779, 28455, 29087, 29676, 30225, 30736, 31210,
    31649, 32054, 32427, 32768, 33079, 33361, 33615, 33842,
    34042, 34217, 34367, 34493, 34595, 34675, 34732, 34767,
    34782, 34776, 34750, 34704, 34639, 34555, 34453, 34333,
    34196, 34041, 33869, 33681, 33476, 33256, 33019, 32768,
    32501, 32220, 31924, 31614, 31290, 30952, 30600, 30235,
    29856, 29465, 29061, 28644, 28215, 27774, 27320, 26855,
    26378, 25889, 25389, 24878, 24356, 23823, 23278, 22724,
    22158, 21582, 20996, 20400, 19794, 19177, 18551, 17915,
    17270, 16615, 15951, 15277, 14595, 13903, 13202, 12493,
    11774, 11047, 10311,  9567,  8814,  8053,  7284,  6506,
     5721,  4927,  4125,  3316,  2498,  1673,   840,     0 };

  for (k = 63; k >= 0; k--) {
    if (k > 7) {
      diff = abs(pixel[k]-pixel[k-8]);
      ++histogram[diff];
      if (max < diff) max = diff;
    }
    if (k & 7) {
      diff = abs(pixel[k]-pixel[k-1]);
      ++histogram[diff];
      if (max < diff) max = diff;
    }
  }
  if (max < 8)
    e = 1 << 16;
  else
    for (k = max; k >= 0; k--)
      if (histogram[k])
        e += AnalysisTable[histogram[k]];
  free(histogram);
  return (e <= ((max + 10) << 10));
}
```

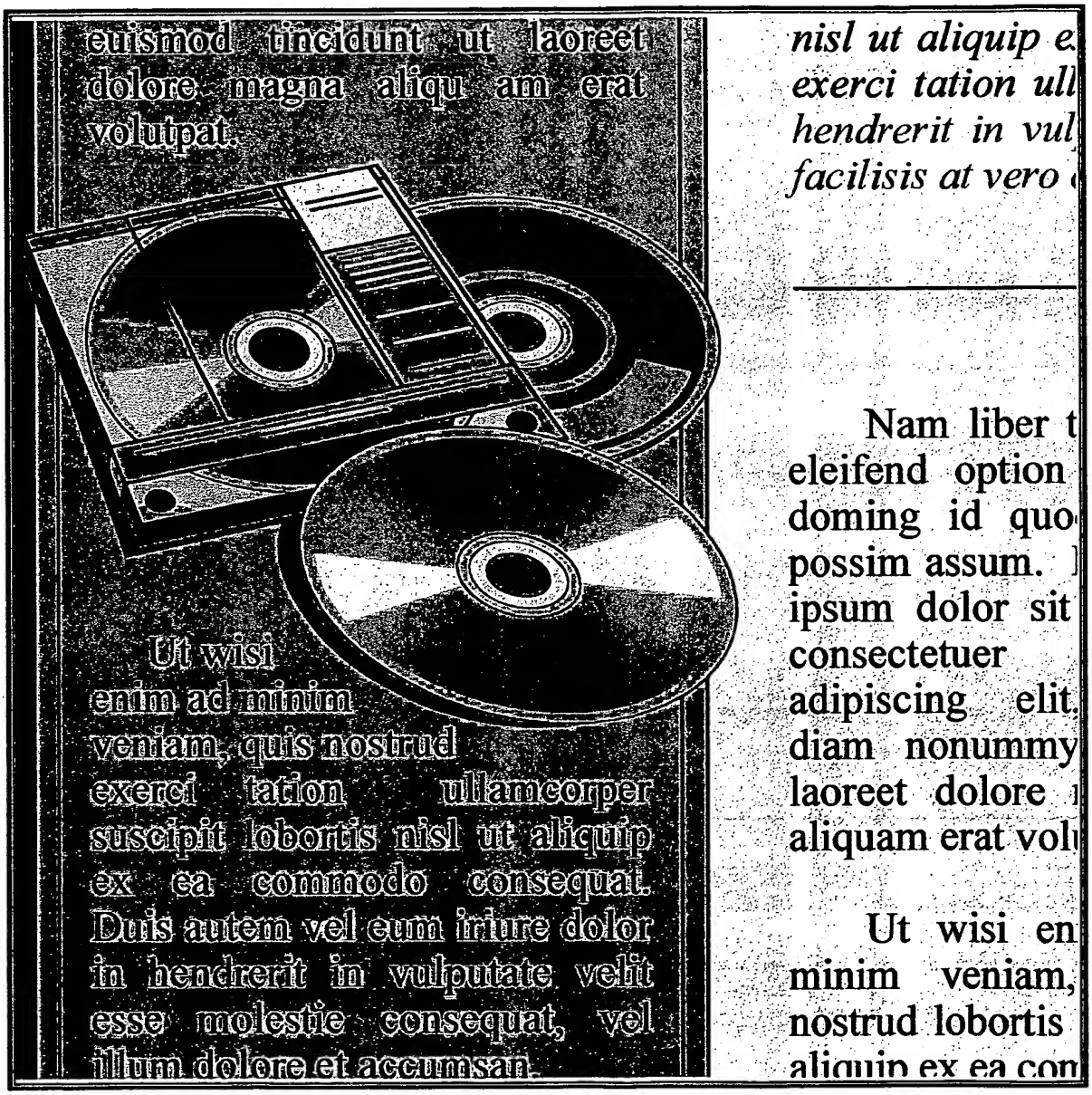


Figure 10 - Classification of the 8×8 blocks based on the entropy of the difference magnitudes. Blocks that have edges or abrupt transitions, and need special coding, are colored amber, all other blocks are colored blue.

Figure 11 - Classification of the 8×8 blocks based on the entropy of the difference magnitudes. Blocks that have edges or abrupt transitions, and need special coding, are colored amber, all other blocks are colored blue.

3.4 Classification Using Compression Efficiency

The entropy function used in the last section is an estimator of the compression that can be achieved in a block. Blocks where the entropy is small can be more effectively compressed, and vice-versa. However, different compression methods do not code different blocks in an equally efficient manner. Their compression is optimal for a given type of image. We can use this property for block classification. If a block is compressed efficiently with a method designed specifically for a given image type, then the block probably is part of a region of that type. For example, if a block is compressed efficient with a method designed for photo compression, then it is probably part of a photo.

We can exploit this fact by sequentially compressing the image blocks with several compression algorithms, and then selecting the algorithm that uses the smallest number of bits. However, a sequential implementation is not very efficient.

Figure 12 shows how a system can exploit this property for simultaneous compression and classification. It is particularly efficient in a parallel hardware implementation. In Figure 12, the pixels in an image block are simultaneously coded using compression methods optimized to different image types. The result of the compression does not go directly to the compressed image stream, instead it is saved in a buffer. When all of the different functions finish compressing the block, a circuit identifies which scheme used the "best" number of bits. Then, only the compressed data chosen as best is copied from its buffer to the compressed stream. The reason why we should not use simply the minimum number of bits for selection will be clear shortly.

An obvious advantage of this classification scheme is that there is no delay while testing the different coding methods. Furthermore, the classification is done *a posteriori*, i.e., there is no delay that would not be related to the compression itself.

As explained before, this type of classification is advantageous only under very specific conditions. Furthermore, there are some issues related to the combination of lossy and lossless methods. Traditional rate-distortion analysis does not apply in this case. However, this is actually true for compound document compression in general, because different image types need different distortion measures. Thus, we have no choice other than rely on heuristic rules to find the best classification. Experimental results show that even simple rules can yield reasonably good compression.

To make the method work it is necessary to define a function for selecting the best bit rate. Choosing the smallest number is not a good choice because lossy methods would have an unfair advantage over lossless methods. The exact solution seems to be extremely complex, but it is easy to find simple empiric rule to avoid such bias. For example, we can multiply the bit rate obtained by a compression method by a constant. These constants should be higher for lossy methods, and lower for lossless methods.

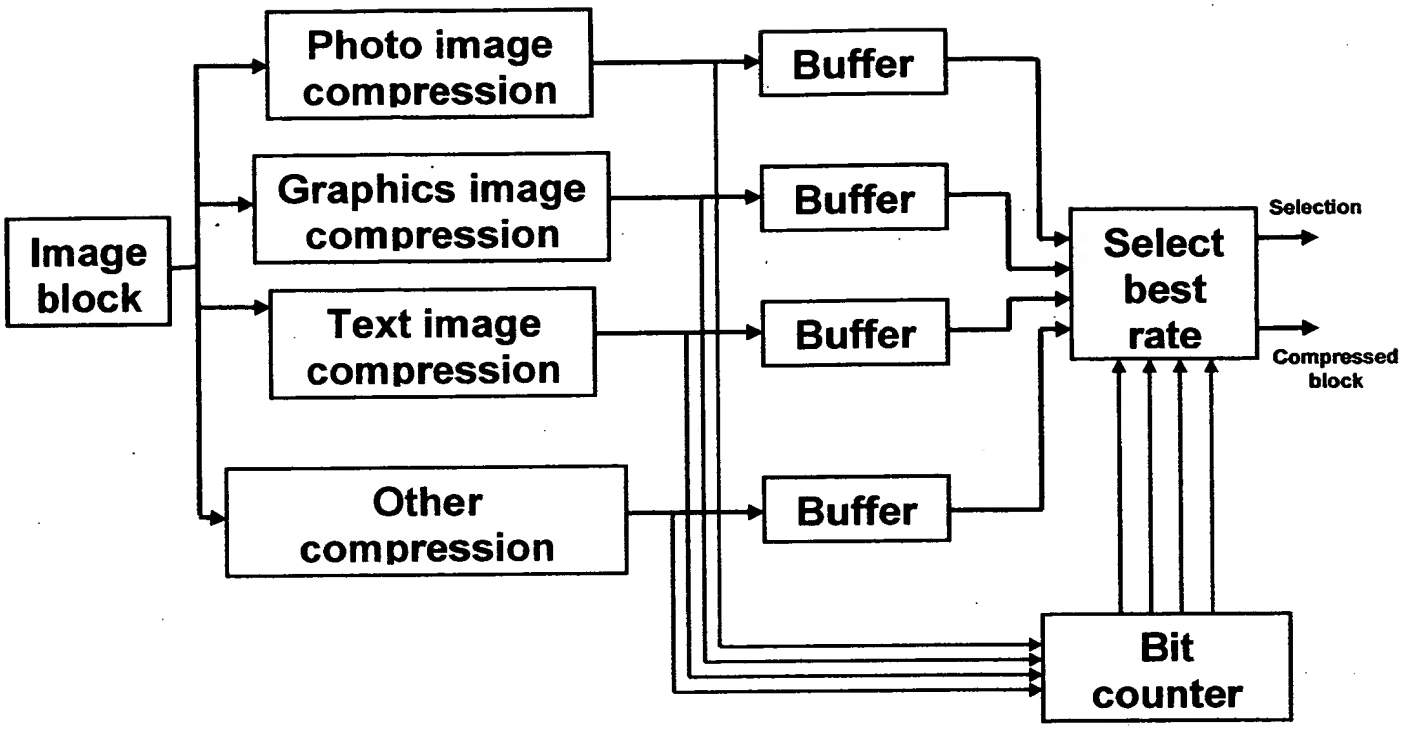


Figure 12 - System for simultaneous compression and classification based on compression efficiency. When using lossy and lossless methods the best rate may not be necessarily the minimum rate.

Experimental results show that such simple rules do work in practice. Figures 13 and 14 show how this compression-efficiency-based classification technique works on the images of Figures 2 and 4. In this experiment we used only two compression methods applied to 8×8 blocks: baseline JPEG (lossy) and a DPCM method (lossless) that used Martucci's MED predictor [9] plus run-length coding of zero residuals. The JPEG bit rate was multiplied by 3 and the DPCM bit rate was left unchanged. Note how the classification correctly identifies text and graphics blocks, and correctly codes them with a lossless compression method.

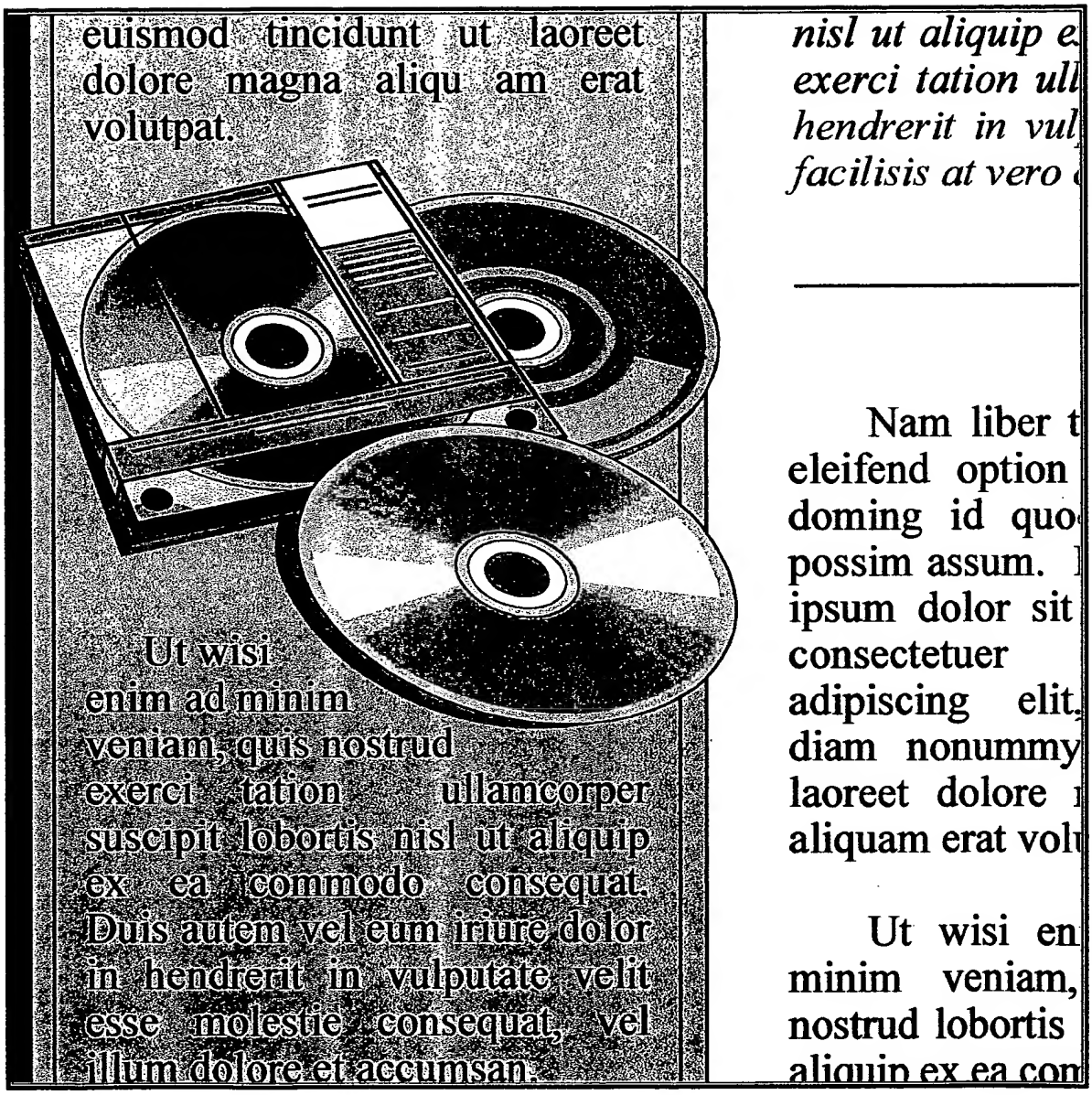


Figure 13 - Classification of 8×8 blocks based on the relative compression efficiency of baseline JPEG and DPCM. Blocks selected for lossy compression with JPEG are colored blue, and blocks selected for lossless compression with DPCM are colored amber.

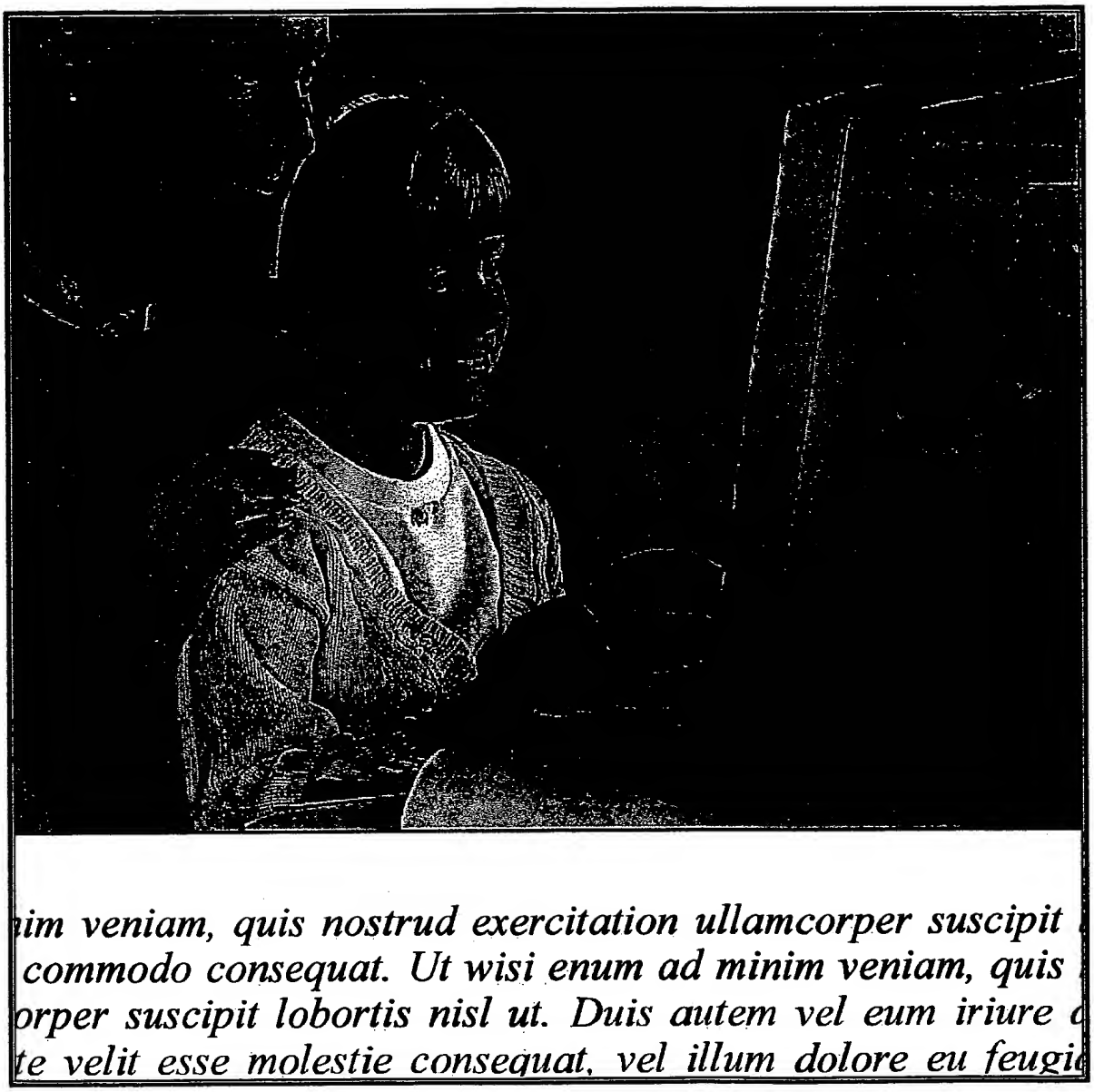


Figure 14 - Classification of 8×8 blocks based on the relative compression efficiency of baseline JPEG and DPCM. Blocks selected for lossy compression with JPEG are colored blue, and blocks selected for lossless compression with DPCM are colored amber.

4 Conclusions

Compound documents can be better compressed if they are first analyzed and its image components are identified. Using this page analysis, the most suited image compression methods are applied to each of the identified component (e.g., text, graphics, photos, background, etc.). An important issue for practical applications is that document analysis can be computationally very expensive. Hard copy devices must process high-resolution documents in real time, with limited resources. Their requirements are so stringent that frequently they can only be satisfied with special hardware or firmware. Under those circumstances, the page analysis and segmentation can only be used if they have minimal complexity.

We propose a set of techniques that allow very fast analysis specifically for compression. They have the following properties.

- *Small memory requirements*: decisions are made on small image blocks (e.g., 8×8 blocks), which fit in small fast-memory buffers (or cache).
- *Small bandwidth*: the pixels in a block are processed only once before deciding on the block class.
- *Small number of operations*: the classification is computed using only a few comparisons and a few simple operations (addition, bit-shifts, etc.) per pixel. A scheme for parallel hardware implementation uses data that has already being computed by the compression method.
- *The classification is matched to compression*: the block classes are meant to be the most useful for compression. For example, for compression of a block it suffices to know that a block has only two distinct colors. It is of less importance to know if it is part of text, of line art, or a graph.

The proposed techniques are classified as:

- *Color analysis*: in typical documents, most of the page analysis can be accomplished by identifying regions with only a few distinct colors. We show that those regions can be identified very efficiently, using at most a few comparisons per pixel. We extend those techniques to deal with noisy data, allowing a tolerance in the color values.
- *Color identification*: we show that colors that occur frequently in a document page can be identified by testing only a small number of pixels randomly selected. We propose an algorithm that uses statistical analysis to minimize the number of pixel tested, and to minimize the number of color vector comparisons.
- *Edge occurrence*: we developed a method to identify regions with edges, and where highly visible lossy compression artifacts. Its purpose is to identify the only the

presence of the edges inside a block, without wasting time identifying the location of the edges. This technique is particularly useful because it complements the color analysis, which cannot always identify the text regions (e.g., when the image is blurred). We developed a discrimination function based on the entropy function applied to a histogram of pixel differences, and based on the maximum difference. This function is not only independent of edge location, but also independent of edge size. Since it analyses distribution, and not particular values, it is also effective for blurred edges.

- *Compression efficiency*: we exploit the fact that each compression methods is most efficient (in a relative manner) for particular image type. Thus, we can measure the compression method's efficiency, and use it to identify the image type. We show that this property can be exploited in a particularly efficient manner using a parallel hardware implementation. We discuss some of the issues regarding the comparison of the relative efficiency of lossy and lossless methods.

All these schemes were actually implemented and tested. The C++ code of some particular implementations is also included in this document. We show the result of applying the proposed method to compound images. The images have the classification shown as a color map, i.e., different colors identify the different classification of the blocks.

The numerical experiments with color identification show that we can identify the predominant colors in an image (or image strip), by testing less than 100 pixels. It is shown that here is practically no loss in performance by testing such small number of pixels: the probability of false-negative and false-positive detection is guaranteed to be less than 1%. The number of color vector comparisons is much smaller than the squared number of pixel tested (brute force approach), and is typically below 200 comparisons.

The application of these classification techniques, and their integration in complete document compression system, is presented in another document.

References

[1] P. Cosman, T. Frajka, D. Schilling, and K. Zeger, "Memory efficient quadtree wavelet coding for compound images," *Research Progress Report for Hewlett-Packard (June 1998-June 1999)*, June 1999.

[2] J. Li and R.M. Gray, "Memory efficient quadtree wavelet coding for compound images," *Proc. IEEE International Conference on Image Processing*, pp. 790–794, Oct. 1998.

[3] K.O. Perlmutter, N. Chaddha, J.B. Buckheit, R.M. Gray, and R.A. Olshen, R.A; "Text segmentation in mixed-mode images using classification trees and transform tree-structured vector quantization," *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 4, pp. 2231 - 2234, May 1996.

[4] M. Pietikäinen and D. Harwood, "Edge information in color images based on histograms of differences," IEEE Comput. Soc. Press (ISBN 0 8186 0742 4), pp. 594–596, Oct. 1986.

[5] N. Chaddha, R. Sharma, A. Agrawal and A. Gupta, "Text segmentation in mixed-mode images," *Rec. Asilomar Conf. Signals, Systems, and Computers*, vol. 2, pp. 1356–1361, Nov. 1994.

[6] T.H. Cormen, C.E. Leiserson, and R.L. Rivest, *Introduction to Algorithms*, The MIT Press, Cambridge, Mass., 1989.

[7] X. Li, W.-G. Oh, S.-Y. Ji, K.-A. Moon,H.-J. Kim, "An efficient method for page segmentation," *Proc. of 1st Int. Conf. on Information Communications and Signal Processing*, pp. 957–961, Singapore, Sept. 1997.

[8] A.K. Jain and B. Yu, "Document representation and its application to page decomposition," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, pp. 294–308, March 1998.

[9] S.A. Martucci, "Reversible compression of HDTV images using median adaptive prediction and arithmetic coding," *Proc. IEEE Int. Symposium on Circuits and Systems*, vol. 2, pp. 1310–1313, May 1990.